Project Report: Quadrotor Planning and Control

Team 21: Zhuheng Jiang, Taylor Shelby, Yihang Xu

I. Introduction and System Overview

The objective of this lab is to test the controllers, trajectory generators and path planners we designed in the earlier phases of this project in a real world environment with real starts, goals and obstacles. This lab also provided an opportunity for us to understand our code from an application perspective and how to modify our code to make a real quadrotor work. The robot capabilities we demonstrated include the stability of the controller, the ability to generate and smooth path in different environments and the ability to generate and follow a feasible and effective trajectory.

The hardware we used was the CrazyFlie 2.0 robot, including an onboard IMU, an onboard controller and the CrazyRadio, a Vicon system, and a PC. Of these, the Vicon system and the onboard IMU provided the sensing capabilities, with the Vicon capturing the position, linear velocity and attitude of the quadroter and the IMU providing angular velocities and accelerations. The computation and control primarily took place in the PC and the onboard controller, with the CrazyRadio acting as the communicator between them. The PC computed the high level commands and produce results based on the Vicon and IMU info, while the onboard controller did low level control and estimation, including the attitude controller. A quaternion representing the desired attitude was sent to the onboard controller from the PC, as was the desired thrust.

II. Controller

The controller we implemented was a geometric nonlinear controller, which attempts to align the thrust of the quadrotor with the desired force developed from the target position, orientation, and their derivatives. This is further explained in the following equations, adapted from [1].

The desired force, in the world frame, is based on the error between the target position r_T and its derivatives, and the current position r and its corresponding derivatives, as shown in Eqn (1). It also includes the force to oppose gravity, using the mass m and gravity g.

$$\boldsymbol{F}^{des} = m(\ddot{r}_T - K_d(\dot{r} - \dot{r}_T) - K_p(r - r_T)) + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{1}$$

K_p and K_d are diagonal positive definite gain matrices. The values used in the lab for these are provided in 2 and 3, respectively.

$$K_p = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 10 \end{bmatrix} \frac{1}{s^2} \tag{2}$$

$$K_d = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6.3 \end{bmatrix} \frac{1}{s} \tag{3}$$

K_d was calculated based on K_p to ensure the system was critically damped in simulation. K_p controls the strength of the system response to position error, while K_d controls the response to velocity error. The desired force is then expressed as a thrust u_1 in the quadrotor frame Z using equation 4, where R represents the orientation of the quadrotor in the inertial frame.

$$u_1 = (R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix})^T \boldsymbol{F}^{des} \tag{4}$$

With the thrust taken care of, it is also necessary to align the orientation of the quadrotor. The desired orientation comes from the desired force and yaw, and is described by equation 5, where b_3^{des} is the unit vector along $\boldsymbol{F}^{des}$ and b_2^{des} is the unit vector along the cross product of b_3^{des} and the vector representing the yaw in the x,y inertial plane.

$$R^{des} = \begin{bmatrix} b_2^{des} \times b_3^{des} & b_2^{des} & b_3^{des} \end{bmatrix} \tag{5}$$

For the lab, this was then simply converted to a quaternion and passed off to the Crazyflie's internal attitude controller. In the simulation, this is instead used to calculate motor speed directly.

A few preemptive changes were made between the simulation and the lab. First, due to the aforementioned onboard attitude controller, the commanded quaternion output had to be set. Likewise, the command thrust had to be output instead of simply providing motor speeds. Beyond changes to match the input required for the lab, we also changed the gains. It is acceptable to take much more aggressive maneuvers in simulation than it is in reality, so we had to scale back the K_p and K_d.

An example of our controller in action can be seen in figure 1. This set of waypoints simply had the quadrotor rise to, and hover at, a Z position of 1.375m. Although the x and y gains are different than the z gains, the relationship between K_p and K_d is the same for all three, so they have the same damping ratio. Fig.

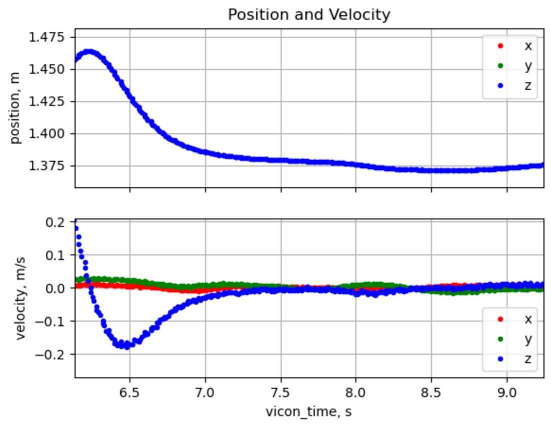

Fig. 1. Step Response in Z direction

1 shows the portion of the path after the quadrotor has followed a constant velocity segment to rise to the desired height. Starting from where the velocity returns to 0, we can see a typical critically damped (or overdamped) response, indicating a damping ratio ≥ 1. The rise time is approximately 0.6 seconds, and the 2% settling time is approximately 1.4 seconds. There is some steady state error as the hovers about the desired height of 1.375m, but it is less than 2 cm. The error for a simple unidirectional step in Z was less than the error for trajectories which also had to move horizontally, since they require the quadrotor to rotate and move, but not immensely so. This is discussed in more detail in section IV.

III. Trajectory Generator

The procedure of getting waypoints had several steps. First, the map was loaded into the program and processed by our A* graph search program which can check the map occupancy and return the best dense path points for the quadrotor if it exists. Then the path was passed to the sparse points function to remove some unnecessary points. The dense path was stored into a list, then a local start point and a checking point (initially the start and the second point) are selected. If the connection between the local start and the checking point has no collision with obstacles, then the check point is updated with next point in the list, and that point is popped from the list. If collision does happen, then the last checking point is set to be the new local start point and the procedure is restarted. All the local start points are stored into a new list to be returned as a sparse path containing only the necessary points. An example is shown in Fig. 4, and the pseudo code for getting the sparse points is attached below.

There was one problem emerged in experiment. When doing the simulation we did not sent all the ”control input” in SE3 controller while in the actual experiments, the thrust, moment and quaternion are required for flight. After adding the extra control inputs, there existed errors when plotting curves in simulation. But the controlling is working for quadrotor. It will not affect the simulation results or the experiments.

```
Q: Dense point path (list) S: Sparse point path (list)
LS: Local start point CP: Checking point
TP: Temporary point
LS ← Q[0]
CP← Q[1]
S[0] ← S[0]
Pop out Q[0]
while Q:
    if LS - CP no collision:
        TP ← CP
        CP← Q[0]
        Pop out Q[0]
    else:
        S + ← TP
        LS ← TP
        CP ← Q[0]
```

In the lab we tested two methods with different types of time allocation, constant velocity and constant acceleration and deceleration bang-bang control. For this part of the report, we will discuss the bang-bang controller, since it is what we tested first. However, we lost the data for the bang bang runs, so the figures later demonstrate and discuss the constant velocity trajectory.

For bang-bang control, the first half accelerates constantly, then at the mid-point the acceleration is 0, then the latter half decelerates constantly. At every one of the sparse way points, the quadrotor comes to a complete stop. This method can easily control the time by setting different acceleration constants, and can constrain the maximum flying speed to prevent the quadrotor from going out of control. First a constant acceleration was set, then using the given points and this acceleration a corresponding time was inferred for each waypoint. For every input t, the corresponding coordinates, velocity and acceleration can be calculated based on this list of times.

Given constant acceleration a, for the segment between i to $i+1$, the time at point i and $i+1$ is t_i and t_{i+1}. At time t in range $[t_i, t_{i+1}]$, the position, velocity and acceleration is:

if $t \leq (t_i + t_{i+1})/2$:

$$x(t) = x_i + \frac{a(t - t_i)^2}{2} \tag{6}$$

$$v(t) = a(t - t_i) \tag{7}$$

$$a(t) = a \tag{8}$$

if $t > (t_i + t_{i+1})/2$:

$$x(t) = x_{i+1} - \frac{a(t_{i+1} - t)^2}{2} \tag{9}$$

$$v(t) = a(t_{i+1} - t) \tag{10}$$

$$a(t) = -a \tag{11}$$

For the constant acceleration bang-bang control we used, the smoothness was not good due to halting at the intermediate points. (Fig. 2) Compared with this method, the constant velocity control we used is much more smooth since it does not stop at every point.(Fig.4) However, this waypoint stop smoothness is not implicitly connected with the feasibility. For the constant acceleration profile, it is relatively simple to select a feasible acceleration. While it is possible the trajectory generator could demand an impossible velocity on a long segment by accelerating for too long, it is simple to check the max velocity of each segment and prevent this. There are discontinuities in the jerk inherent to this method, so it is technically infeasible, which results in the simulation velocity error seen in Fig. 3. As long as the acceleration and velocity are not extremely large the PD controller can handle this despite the infeasiblity. The same general idea applies to the constant velocity trajectory, which has discontinuities in the acceleration, but since this trajectory did not stop at each waypoint it ended up smoother overall.(Figs. 3,4)

IV. Maze Flight Experiments

Using map 1 as an example (Figs. 4, 5), we can see how the constant velocity trajectory fared in the actual lab. The constant velocity was very slow, even when we changed the code, but this made it so the controller was able to keep the quadrotor fairly close to the desired trajectory. (Fig. 8) The trajectory could have been even closer to the obstacles if we knew it was going to be so slow. With the margin we had, it could also have moved a lot faster and still been safe. As shown in figures 8 - 10, we can tell that the average tracking error is around 0.1 and the largest error is around 0.2. We can use the tracking error to adjust the margin of the obstacles in the A* planner. This helps make sure that the quadrotor can find a path through the obstacles while it won't collide with them. Based on our error, the margin can be set around 0.25-0.30 and still be safely outside the max error we saw in the lab.

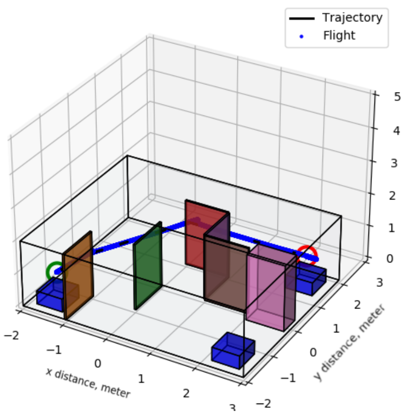

Fig. 2. 3-D flight path in maze 1 simulation, Bang Bang Acceleration

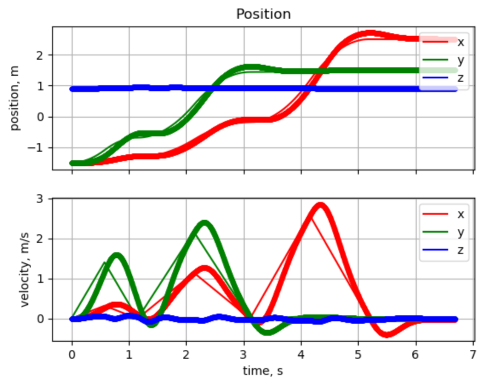

Fig. 3. Position and velocity in maze 1 simulation, Bang Bang Acceleration

To make sure that the quadrotor flew safely, we used a very slow speed in the lab. In the simulation, we ran the speed up to 2 to 3 m/s without collision, so there is definitely room for increasing the aggressiveness of the trajectory.

For both trajectory generators we tried in the lab, a better controller would enable the quadrotor to be more stable in aggressive moves (high-speed moving and sharp curves). This would be particularly helpful on the bang-bang acceleration trajectory, since even in simulation it deviated fairly significantly from the desired values. (Fig. 3) For the constant velocity trajectory, simply increasing the selected velocity would have increased the speed, but we had difficulties getting it to work in the lab. For the bang-bang trajectory which stopped at every waypoint, an algorithm like Douglas-Peuker could have been used to further reduce the number of points, reducing the time for that path.

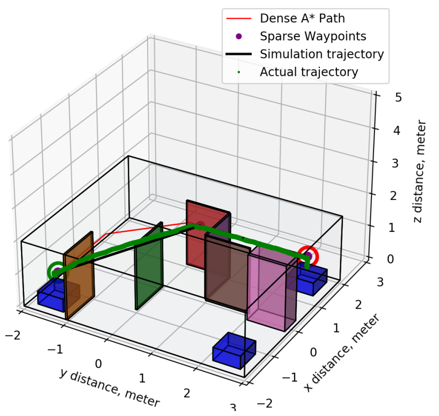

Fig. 4. 3D maze 1, constant velocity trajectory

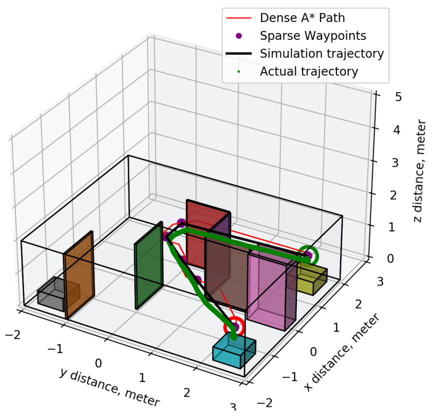

Fig. 6. 3D maze 2, constant velocity trajectory

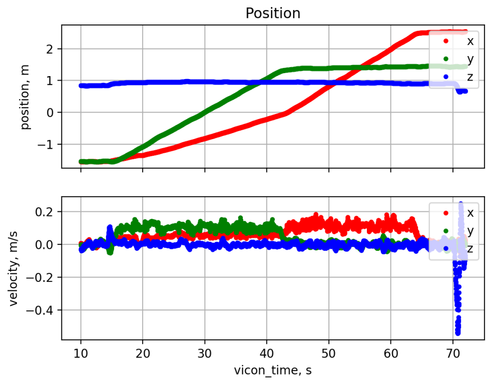

Fig. 5. Position and velocity vs time of actual flight in maze 1

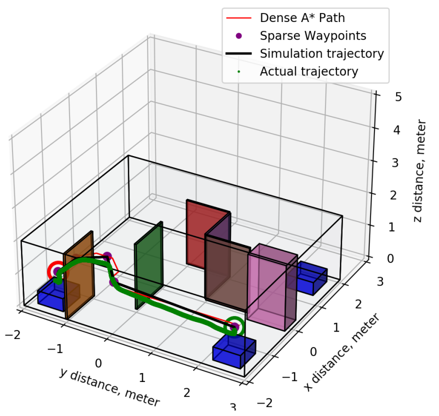

Fig. 7. 3D maze 3, constant velocity trajectory

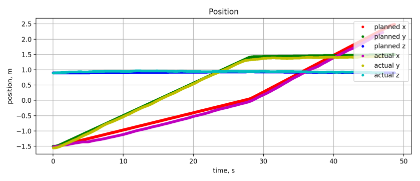

Fig. 8. Error between planned and actual trajectory of maze1

Beyond these adjustments to the systems we had, there are other trajectories which would have allowed faster and more reliable results in lab. For example, using a min-snap quadratic program with corridor and continuity constraints guarantees a safe, feasible trajectory.

If we had one more session, we would want to test another trajectory method, such the fully constrained minimum jerk trajectory or minimum snap with corridor constraints. Between the three of us, after the lab we finished both a min-jerk method and a min-snap with corridor constraints method, so it would be fun to test these in the lab.

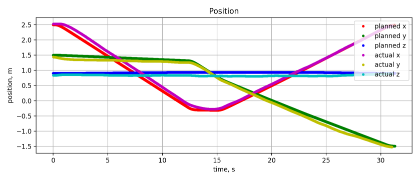

Fig. 9. Error between planned and actual trajectory of maze2

References

[1] MEAM 620 Project 1 Phase 1, James Paulos
[2] http://engineering.ju.edu.jo/Laboratories/05%20-%20Second%20Order%20System%20and%20Higher%20order.pdf

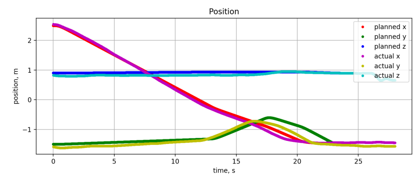

Fig. 10. Error between planned and actual trajectory of maze3